3D Gaussian beam modeling

Paul Hursky
3366 North Torrey Pines Court, Suite 310
La Jolla, CA 92037
phone: (858) 457-0800 fax: (858) 457-0801 email: paul.hursky@hlsresearch.com

Award Number: N00014-09-C-0425
http://hlsresearch.com

LONG-TERM GOALS

Long-term goals of this research are:

- provide the underwater acoustic research community and the US Navy with a practical 3D propagation model that can be used over a broad frequency range in passive and active sonar performance prediction, acoustic communications channel modeling, and in understanding operational concepts impacted by 3D propagation effects.
- provide these predictive capabilities for both pressure and particle velocity sensors.

OBJECTIVES

The initial objective was to implement a 3D Gaussian beam model on multi-core, high-performance computing hardware in order to support mid and high frequency applications like reverberation modeling and acoustic channel simulation. As graphic processing unit or GPU hardware rapidly emerged as a compelling platform for such work, my objective became to adapt selected acoustic wave propagation models onto GPUs.

APPROACH

Assessing the availability and maturity of available numerical building blocks for GPUs, I chose a split-step Fourier parabolic equation model for an initial objective – this would allow me to assess the potential of this technology, produce a useful product, and at the same time lay the groundwork for developing a GPU-based 3D Gaussian beam model. There was a very successful off-the-shelf FFT implementation for GPUs. Linear algebra tools needed for some of the other models (e.g. tri-diagonal linear solver, needed by split-step Pade PE or wavenumber integral models) were still works in progress. I selected the split-step Fourier PE (SSFPE) model for GPU implementation. I will refer to this implementation as GPU-SSFPE. Although this particular PE model has been somewhat eclipsed by the split-step Pade PE (which requires a tri-diagonal solver), it is still part of the standard OAML set of models, and a case can be made for its use in deep water cases.

NVIDIA was the GPU manufacturer that had been most supportive of high performance scientific computing (in terms of providing tools and software libraries). NVIDIA provides the "Compute

Report Documentation Page		*Form Approved OMB No. 0704-0188*

1. REPORT DATE **SEP 2011**	2. REPORT TYPE	3. DATES COVERED **00-00-2011 to 00-00-2011**
4. TITLE AND SUBTITLE **3D Gaussian beam modeling**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **HLS Research,3366 North Torrey Pines Court, Suite 310,La Jolla,CA,92037**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**11**	

Unified Device Architecture" or CUDA framework for developing software to run on GPUs. CUDA provides a C language interface to the GPU hardware. CUDA entails programming in C with a few added programming constructs to set up "Single Instruction Multiple Thread" or SIMT programs called "kernels" which execute in parallel on the individual GPU cores. The strategy for implementing significantly accelerated GPU applications requires managing the flow of data between the CPU and the GPU, and organizing the flow of data between several specialized memory units, such as global memory, shared memory, constant memory, texture memory, and register memory. There are ample opportunities to set up parallelism of the various data transfers with computations on both the CPU and the GPU. Typically, the bottleneck is in transferring data in and out of the GPU – if too little arithmetic is required in the application for the data being moved between the CPU and the GPU, the GPU cores will not be kept busy enough to achieve the desired acceleration.

I discovered that NVIDIA had opened up their OptiX software infrastructure for ray tracing (tracing optical rays in order render or display complex 3D scenes) to scientific applications, like using rays to model sound propagation. To use OptiX for underwater sound propagation, the 3D waveguide structure must be represented using the data structures used by OptiX for ray tracing. However, it is possible to modify the ray "payload" and "interactions" (with "objects" like a sound speed layer) to represent customized models (like a Gaussian beam model for underwater acoustics).

There is a great deal of commonality between a 3D Gaussian beam model for underwater acoustics and ray tracing of light rays in order to "render" realistic 3D "scenes", capturing effects such as caustics caused by refraction and transmission through glass objects for example. Ray tracing is not trivially parallelized. Different rays encounter different objects and are "extinguished" at different path lengths, so the computational load is not evenly distributed over rays. Nevertheless, ray tracing is of sufficient importance to the computer graphics industry that considerable effort has been spent on achieving real-time, high-definition ray traced rendering. As a result, various "acceleration structures" have been developed in order to run ray tracing efficiently on parallel architectures. NVIDIA, the GPU company that has been most aggressive in supporting high-performance computing and scientific computing on GPUs, provides a general-purpose ray-tracing framework (called OptiX) that allows third party developers (like us) to incorporate their own "ray payload" and their own mathematics into all the ray interactions with objects in a 3D model. In our case, the "ray payload" include all of the Gaussian beam parameters that are numerically integrated along the rays, and the mathematics consists of our particular algorithms for numerically integrating these parameters based on the medium properties, which are incorporated into a 3D model of our ocean waveguide. The mathematics is programmed using CUDA C, the same "platform" I have used to implement our split-step Fourier PE model.

WORK COMPLETED

An important application for a Gaussian beam model, and perhaps the most demanding in terms of computational load, is synthesizing a high-frequency acoustic channel, for acoustic communications. To lay the groundwork for this application, in particular for vector sensors, I did some prototyping and analysis, using the Matlab version of Bellhop and the VirTEX channel simulator that was developed at HLS Research by Martin Siderius and Michael Porter. Bellhop provides a choice of several "influence functions" for calculating the field. I adapted several of these beam influence functions to produce particle velocity and benchmarked them against the wavenumber-integral OASES model developed by Henrik Schmidt (which can also calculate particle velocities). Integrating these influence functions into a channel simulator involves convolving the analytic signal form of the broadband acomms waveform (being transmitted through the channel) with the channel impulse response. A simple and efficient way to perform this operation in terms of the Gaussian beam constructs is to recognize that the pressure gradient along the ray tangent is equivalent to the time derivative of the analytic signal being convolved, while the pressure gradient along the ray normal is the derivative with respect to the influence function. These two orthogonal convolved components (i.e. ray tangent and ray normal for each traced ray or beam) are projected onto the range and depth axes to form the vector sensor broadband output. This work was completed prior to my having set a course to work with GPUs.

I assessed evolving "General Purpose Graphic Processing Unit" (GPGPU) developments, particularly key low-level building blocks needed to implement the various types of acoustic propagation models, such as FFTs and linear algebra building blocks in BLAS and LAPACK. I attended several tutorial sessions on GPGPU software development. I developed an understanding of the underlying hardware on GPUs and reviewed the available software tools for developing applications. Basically, of the two major GPU manufacturers, NVIDIA and ATI (now part of AMD), NVIDIA was aggressively evangelizing use of GPUs in high performance scientific computing, essentially by providing free software development tools and fostering the development of generally useful algorithmic building blocks (like FFTs and BLAS/LAPACK).

There are a number of high-level tools that attempt to hide some of this complexity, but my experimentation with these has not yielded good results. There is no interface for specifying how the data is to flow among the GPU and its specialized memories. Furthermore, it is hard to find a full repertoire of the mathematical functions needed, or, if they exist, they may run four times slower than their CPU counterparts. There are also alternative language bindings to CUDA, including Python (PyCUDA) and Fortran (from Portland Group, which has both a low-level FORTRAN interface to CUDA, and an "Accelerator" which tries to convert existing Fortran code to parallel form to be run in CUDA). A Fortran binding may be helpful for porting a model that is already coded in Fortran, but it is hard to imagine how this can be successful, without intelligent restructuring of the original code to adapt it to the GPU architecture. Although I was a bit apprehensive about coding in C, I think that this has allowed me to best expose the potential of this technology. It also set the stage for the more ambitious Gaussian beam implementation using OptiX, which interfaces with CUDA C.

In an initial version of GPU-SSFPE, the GPU was used purely as a FFT engine, in order to establish the performance boost that could be achieved on GPUs, and to assess the level of difficulty for developing a more complete GPU implementation. This initial foray into GPU programming produced speedups of 10 to 20 times relative to a CPU-only implementation, and was not significantly more difficult to program than a CPU-only version, in which FFTW ("fastest FFT in the West") was used as an FFT engine. The open-source FFTW is widely used, including by Matlab, and is reported to be only slightly slower than commercial libraries such as Intel MKL FFT. The FFTW implementation also has a multi-core CPU version using OpenMP, an open-source software standard for parallelizing multiple cores communicating through a shared memory. The OpenMP version of FFTW did not yield run times that scaled with the number of cores, so perhaps there is a better multi-threaded FFT (e.g. in Intel's MKL library).

Once I convinced myself that the performance gains through GPUs were significant, I migrated all the mathematics onto the GPU and fleshed out a relatively complete set of features that would be expected in a split-step Fourier PE. This included three variants of the operator splitting, including $e^{A\Delta r}e^{B\Delta r}$ (case I in Ref. [14]), $e^{\frac{A}{2}\Delta r}e^{B\Delta r}e^{\frac{A}{2}\Delta r}$(case III in [14]), and $e^{\frac{B}{2}\Delta r}e^{A\Delta r}e^{\frac{B}{2}\Delta r}$ (a wider-angle form from [15], that provides the best accuracy in the split-step Fourier repertoire). Each of these operator splittings has different forms for the A and B operators. I implemented the following starter fields: Gaussian starter, Greene starter, normal mode starter, Thomson starter (see [17]), the self-starter (see [16]), and a user-provided starter field read from a file. My implementation uses a reduced pressure formulation in which pressure $\tilde{p}(r,z) = \frac{p(r,z)}{\sqrt{\rho}}$ is stepped in range and then multiplied by the square root of the density before being output. Using this reduced pressure form enables the density change at the water-seabed interface to be handled properly. The model inputs are provided through an ASCII input spec that is based on the RAM input spec (RAM can be run off of this spec), but expanded to specify the various starter fields. The operator splitting and the number of threads to use on the GPU are specified using a command line argument. The model outputs a complex pressure field, and, optionally, all of the intermediate results generated in each range step, if needed (for debugging purposes).

The current implementation assumes the sound speed is fixed and the bathymetry is smoothly varying. The GPU-friendly way to accommodate range-dependent sound speed is to download a sparse representation of it into the GPU and to use GPU-resident interpolation to produce the finely sampled profiles needed by the model as it marches out in range. I was initially planning to do this by loading the sound speed into texture memory, and then to use the hardware-accelerated interpolation provided by the texture memory. All the waveguide specs can be handled in a similar manner. However, the most recent generation of the NVIDIA hardware has features which reduce the advantage of using the texture memory. As a result, I have set this enhancement aside until a clearer consensus emerges. Currently, all the i/o between the host and the GPU is “hidden” by doing GPU calculations, host calculations, and the data transfers all in parallel.

I benchmarked GPU-SSFPE against historical benchmarks. In making these comparisons I was able to duplicate the historical results, even to a fault (e.g. the leaky duct problem encountered by the Thomson-Chapman formulation, that was fixed by using a different c0). However, I discovered a discrepancy that is fundamental to the SSFPE for problems with significant density and sound speed contrasts. For an ocean-seabed boundary with cp_water=1500, cp_seabed of 1700, and density ratio of 1.5, results at long range (>15km) in shallow water exhibited a shift in range of the TL pattern, when compared with either RAM or Scooter. Kevin Smith graciously provided the source code for the MMPE model, which is also a split-step Fourier model. Comparisons with MMPE on this and other cases, confirmed that this phase offset was endemic to the split-step Fourier PE. Kevin and I found that the discrepancy could be “fixed” by slightly increasing the waveguide depth, but I have not found a pre-scriptive solution, which can be applied without having a benchmark solution in hand already. My hypothesis is that the phase offset is caused by the smoothing function applied to the density contrast at the ocean-seabed interface, so I have been using other models with the same artificial smoothing function added to the medium in order to understand how this discrepancy can be overcome.

RESULTS

I have learned how to adapt algorithms implemented on single and multi-core CPUs to GPUs. As optimized building blocks such as a parallelized tri-diagonal linear solver (used in the “implicit finite differences” and split-step Pade PE models, as well as in the wavenumber integral models) become available, I will be able to produce a unified suite of PE models, all running on GPUs, and using the same input spec and output formats.

Timing comparisons were performed on three range-dependent cases shown below, which I have labeled as: case m5rd, a Munk profile at 400 Hz in which the water depth decreases smoothly from 5 km to 4.5 km; case dickens, with a shallow duct and a 3 km water column depth, interrupted by a seamount, at 230 Hz; and case w500, modeling upslope propagation in a wedge at 500 Hz.

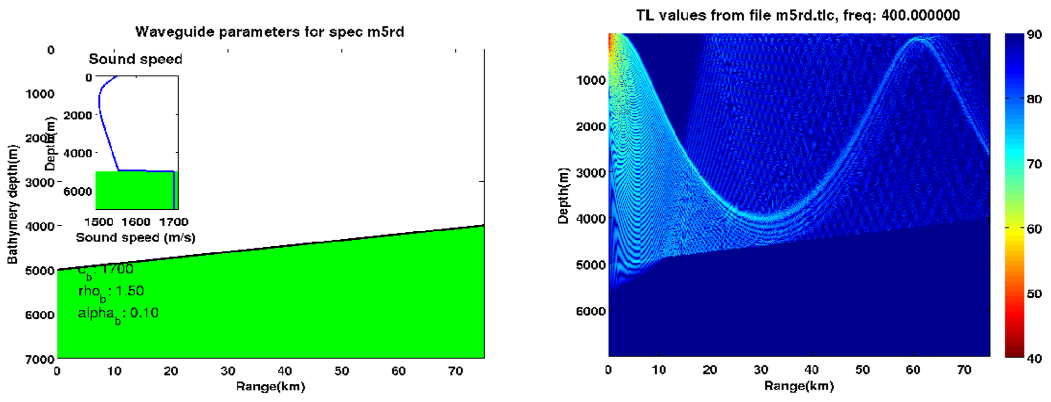

Figure 1: Case m5rd, range-dependent Munk profile, source at 150 m depth, at 400 Hz, zmax=7000, dz=.854492, and fft_sz=16k.

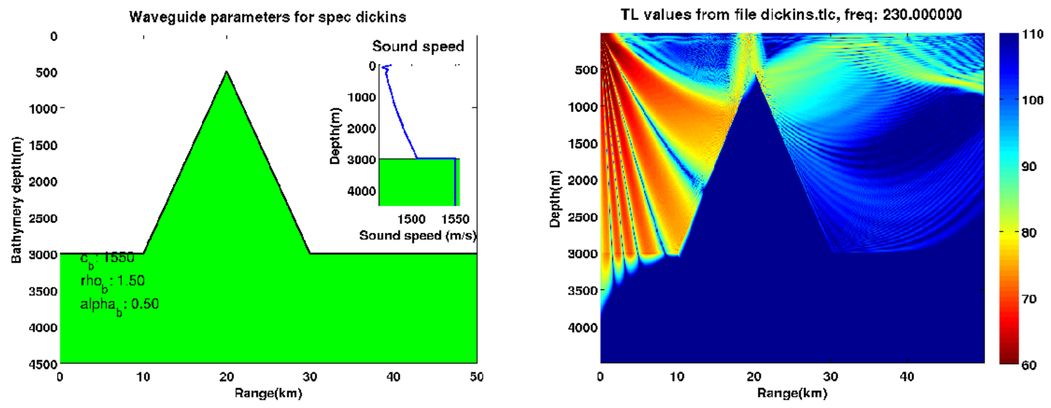

Figure 2: Case dickins, Dickins seamount, source at 18 m depth, at 230 Hz, zmax=4500, dz=.54931640625, and fft_sz=16k.

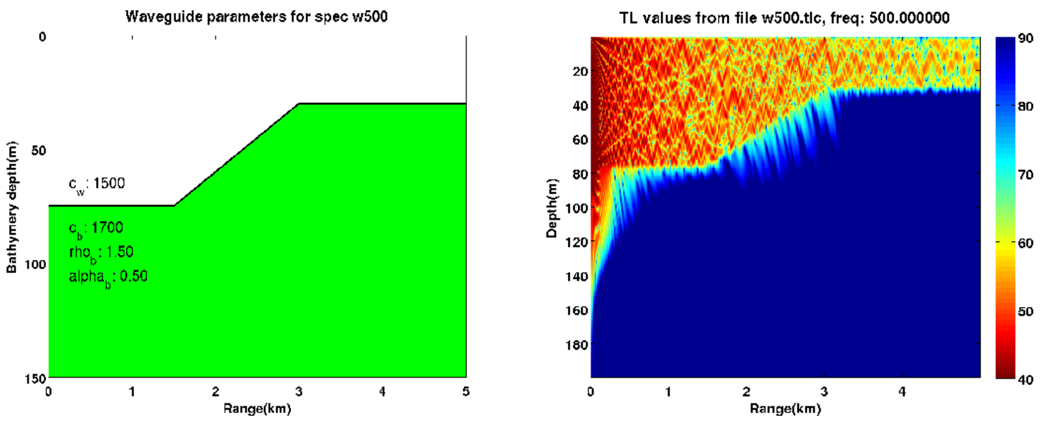

Figure 3: Case w500, upslope propagation in an isovelocity wedge, source at 40 m depth, at 500 Hz, zmax=200, dz=.0625, and fft_sz=6.4k.

The following tables show the results of running the CPU and GPU versions of the three cases described above with varying numbers of cores or threads. The four systems on which the timing comparisons were performed have CPUs and GPUs with varying capabilities and vintages. The GFLOPs/sec numbers do not include all the calculations, only those associated with applying the two operators, as described above, so these numbers will be underestimates. Nevertheless, these measurements can be used to compare the CPU and GPU capabilities.

Table 1: Laptop, Intel Core i7 Q 720 1.6 GHz, Quad core (4 cores, 8 threads), Q3 '09, GeForce GTS 360M (previous generation, mobile version), 96 cores, 1.32 GHz, CUDA 4.0.

	Seconds		GFLOPs/sec		Ratio	No. threads	
	GPU	CPU	GPU	CPU		GPU	CPU
m5rd	36.19	486.57	10.57	0.79	13.44	128	4
	36.01	529.34	10.63	0.72	14.70	256	8
dickins	10.65	91.34	11.23	1.31	8.58	128	4
	10.6	104.82	11.28	1.14	9.89	256	8
w500	8.02	25.17	4.23	1.35	3.14	128	4
	8.03	25.45	4.22	1.33	3.17	256	8

Table 2: Desktop, Intel Core 2 Q9550 2.83 GHz Quad core (4 cores, 4 threads), Q1 '08, GeForce GTX 460 (current generation), 336 cores, 1.35 GHz, CUDA 4.0.

	Seconds		GFLOPs/sec		Ratio	No. threads	
	GPU	CPU	GPU	CPU		GPU	CPU
m5rd	9.66	338.34	39.61	1.13	35.01	128	4
	9.57	360.56	40.00	1.06	37.68	256	8
dickins	2.77	61.73	43.14	1.94	22.26	128	4
	2.76	67.98	43.31	1.76	24.62	256	8
w500	2.94	16.95	11.52	2.00	5.76	128	4
	2.94	18.06	11.52	1.88	6.13	256	8

Table 3: Desktop, Intel Core i7 950 3.07 GHz (4 cores, 8 threads), Q2 '09, GeForce GTX 460 (current generation), 336 cores, 1.35 GHz, CUDA 4.0.

	Seconds		GFLOPs/sec		Ratio	No. threads	
	GPU	CPU	GPU	CPU		GPU	CPU
m5rd	9.34	310.89	40.98	1.23	33.29	128	4
	9.25	336.18	41.37	1.14	36.34	256	8
dickins	2.71	57	44.20	2.10	21.06	128	4
	2.69	64.93	44.39	1.84	24.10	256	8
w500	2.87	15.64	11.80	2.17	5.44	128	4
	2.87	15.96	11.81	2.12	5.56	256	8

Table 4: Server, dual-slot Intel Xeon X5550 2.67 GHz, quad core (8 cores, 16 threads), Q1 '09, GeForce GTX 285 (prev. generation), 240 cores, 1.48 GHz, CUDA 3.2.

	Seconds		GFLOPs/sec		Ratio	No. threads	
	GPU	CPU	GPU	CPU		GPU	CPU
m5rd	20.63	302.02	18.55	1.27	14.64	128	4
	20.66	315.26	18.53	1.21	15.26	256	8
dickins	6.2	56.91	19.29	2.10	9.18	128	4
	6.21	61.4	19.26	1.95	9.89	256	8
w500	6.45	16.23	5.26	2.09	2.52	128	4
	6.47	15.27	5.24	2.22	2.36	256	8

I have produced an accelerated SSFPE model, running 10-40 times faster than a comparable CPU-based version, that can be used as a drop-in replacement for CPU-based versions, including UMPE/MMPE and RAM. Both versions were coded in C, using nearly identical code for the mathematics, except for the calls to CUDA FFTs on the GPU and calls to FFTW on the CPU. A factor of 10-40 times represents a significant amount of acceleration, especially considering that the cost of the hardware required to realize it is readily available in off-the-shelf laptops, desktops and servers, is updated more frequently than most CPUs (every 6-9 months), and costs significantly less than a typical computer desktop or laptop ($250-$500). Higher-end models with up to 6 GB of RAM, full double-precision capability, and ECC RAM costing more than $2000 are available.

To assess the correctness of GPU-SSFPE, I have compared GPU-SSFPE outputs with the outputs of Scooter (see[18]), a wavenumber integration model, and the outputs of RAM, a split-step Pade PE model (see [4]).

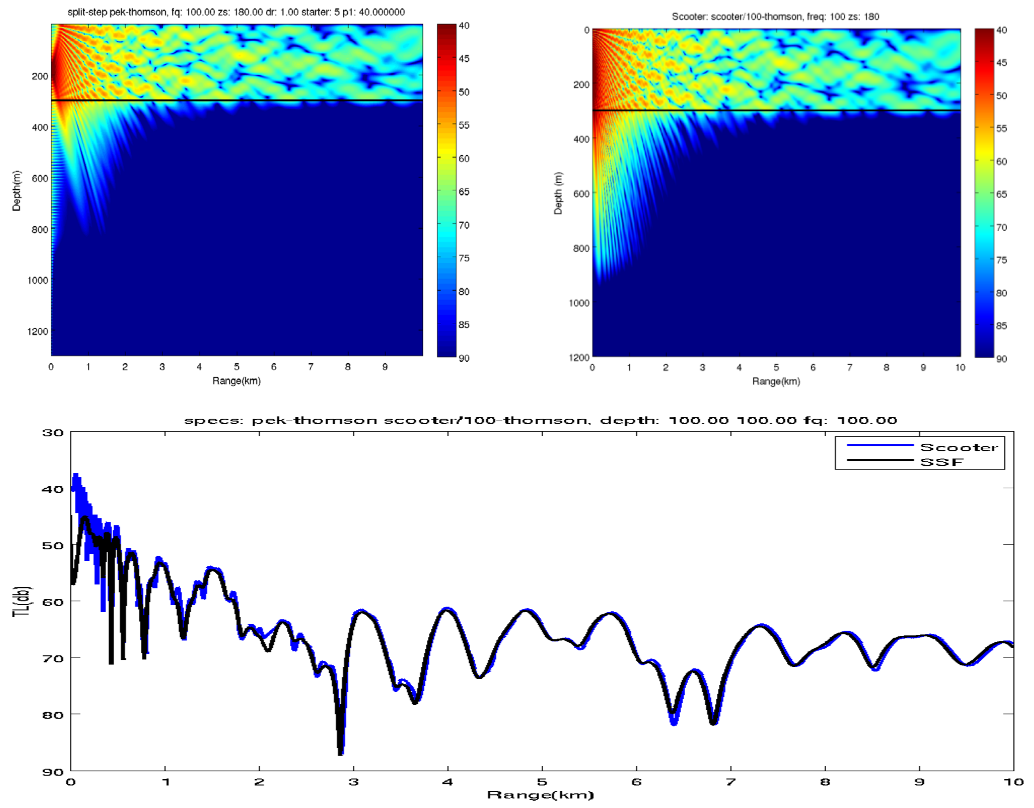

Figure 4: Pekeris waveguide comparison at 100 Hz between Scooter and our split-step Fourier PE, with sound speeds of 1500 m/s (water) and 1550 m/s (seabed), seabed density 1.5, and attenuation .5 db/wavelength, 300 m water depth, source depth of 180 m, with line plot TL comparison for receiver at depth 100 m.

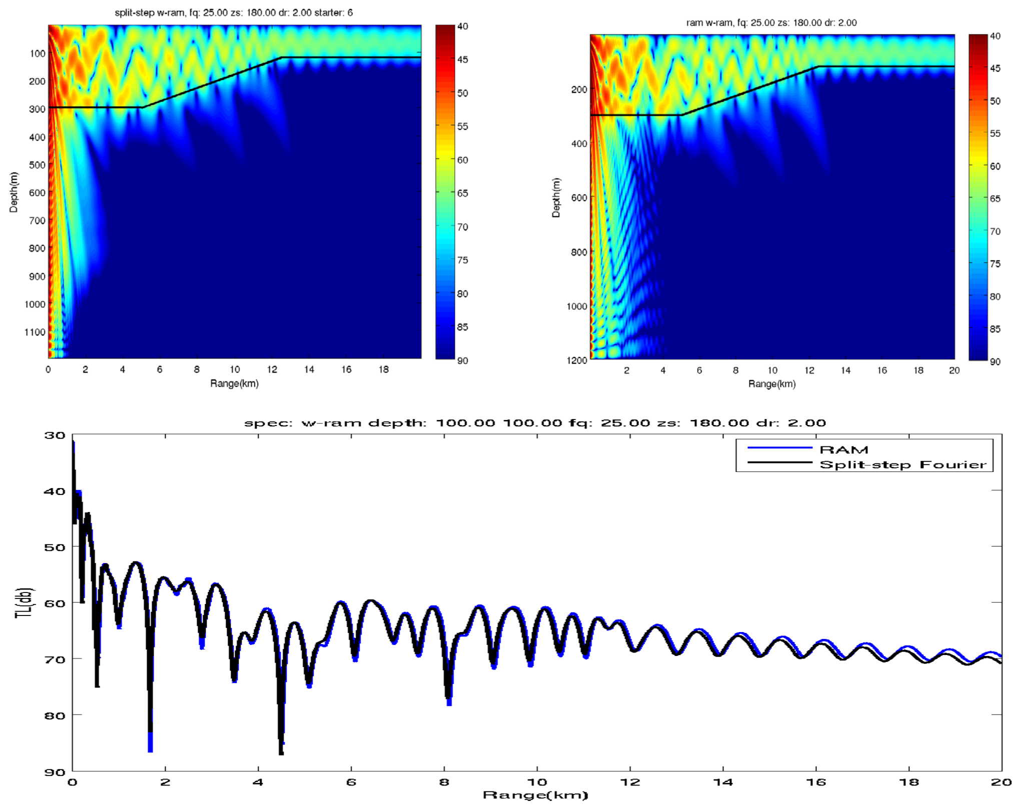

Figure 5: Wedge comparison at 25 Hz between RAM and our split-step Fourier PE, with sound speeds of 1500 m/s (water) and 1700 m/s (seabed), no density contrast at the interface, and attenuation .5 db/wavelength, with water depth starting out at 300 m, then decreasing from 300 m to 120 m from range of 5 km to 12.5 km (a slope of 1.3 degrees), and source depth of 180 m, with line plot TL comparison for receiver at depth 100 m.

IMPACT/APPLICATIONS

The impact of GPU technology on high performance computing is clear. Clusters of computing nodes with multi-core CPUs are being augmented with GPUs. For example, a 4U dual-slot server can be outfitted with eight GPU cards to greatly increase the GFLOPs per volume and per watt. Interestingly, GPUs are being considered for embedded applications as well, because the GFLOPs per watt of GPUs is competitive. To realize the potential benefits of this technology, it will be important to adapt existing and emerging acoustic models (e.g. 3D models) to the GPU architecture.

RELATED PROJECTS

I have presented these results at the HIFAST conference and we are hoping to obtain funding to accelerate reverberation and target scattering models using GPU technology.

REFERENCES

1. Steinar H. Gunderson. GPUwave, http://gpuwave.sesse.net/, 2007 (an earlier GPU implementation, using low-level “shader” functions, prior to availability of CUDA).
2. Suzanne T. McDaniel and Ding Lee. “A finite-difference treatment of interface conditions for the parabolic wave equation: The horizontal interface.” The Journal of the Acoustical Society of America, 71(4):855, 1982.
3. Ding Lee and Suzanne T. McDaniel. “A finite-difference treatment of interface conditions for the parabolic wave equation: The irregular interface.” The Journal of the Acoustical Society of America, 73(5):1441, 1983.
4. Michael D. Collins. “A split-step Pade solution for the parabolic equation method.” The Journal of the Acoustical Society of America, 93(4):1736, 1993.
5. Michael D. Collins. “Generalization of the split-step Pade solution.” The Journal of the Acoustical Society of America, 96(1):382, 1994.
6. Jonathan Cohen and John D Owens and Yao Zhang. “Fast tridiagonal solvers on the GPU.” Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming PPoPP 10, pages 127, New York, NY, 2010. ACM.
7. M. Frigo and S.G. Johnson. “The Design and Implementation of FFTW3.” Proceedings of the IEEE, 93(2):216-231, 2005.
8. Jason Sanders and Edward Kandrot. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, 2010.
9. David B. Kirk and Wen-mei W. Hwu. Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann, 2010.
10. NVIDIA CUDA C Programming Guide. NVIDIA, v4.0 edition, 2011.
11. CUDA C Best Practices Guide. NVIDIA, v4.0 edition, 2011.
12. CUDA CUFFT Library. NVIDIA, v1.0 edition, 2011.
13. Frederick Tappert. “The parabolic approximation method.” Wave Propagation in Underwater Acoustics, edited by J.B. Keller and J.S. Papadakis in Lecture Notes in Physics, pages 224-287. Springer-Verlag, New York, NY, 1977.
14. Finn B. Jensen and William A. Kuperman and Michael B. Porter and Henrik Schmidt. Computational Ocean Acoustics. Springer-Verlag, New York, NY, 1997.
15. D. J. Thomson and N.R. Chapman. “A wide-angle split-step algorithm for the parabolic equation.” The Journal of the Acoustical Society of America, 74(6):1848, 1983.
16. Michael D. Collins. “A self-starter for the parabolic equation method”, The Journal of the Acoustical Society of America, 92(4):2069, 1992.
17. David J. Thomson, “Wide-angle parabolic equation solutions to two range-dependent benchmark problems”, The Journal of the Acoustical Society of America, 87(4):1514, 1990.
18. Michael B. Porter, The Acoustic Toolbox, http://oalib.hlsresearch.com/Modes/AcousticToolbox (this modeling suite contains the Scooter wavenumber integration model used as a benchmark for our split-step Fourier PE model).

19. Michael D. Collins, Users guide for RAM, versions 1.0 and 1.0p.

PUBLICATIONS

1. Kai Tu, Dario Fertonani, Tolga M. Duman, Milica Stojanovic, John G. Proakis and Paul Hursky, “Mitigation of Intercarrrier Interference for OFDM for Time-Varying Underwater Acoustic Channels,” IEEE J. Ocean. Eng. 36(1):156-171 (2011).

2. Aijun Song; Abdi, A.; Badiey, M.; Hursky, P., "Experimental Demonstration of Underwater Acoustic Communication by Vector Sensors," *Oceanic Engineering, IEEE Journal of* , vol.36, no.3, pp.454-461, July 2011.

3. Paul Hursky and Michael B. Porter, “Implementation of a split-step Fourier Parabolic Equation model on graphic processing units”, 4th Underwater Acoustics Measurements Conference, UAM 2011, 20-24 June 2011, Kos, Greece.

4. Paul Hursky, Ahmad T. Abawi, and Michael B. Porter, “Benefit of two-dimensional aperture in matched-field processing”, 4th Underwater Acoustics Measurements Conference, UAM 2011, 20-24 June 2011, Kos, Greece.

5. Paul Hursky and Michael B. Porter, “Accelerating Underwater Acoustic Propagation Modeling Using General Purpose Graphic Processing Units”, Oceans ‘11 IEEE/MTS Kona, Kona, Hawaii, 19-22 September 2011.